Homework 1

Shiva Kalyan Sunder Diwakaruni

CPSC 8430 – DEEP LEARNING

HW1_1_1 Simulate a function

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_1_1.ipynb

Target Function

The target function used in these experiments combines a sine function with a logarithmic function, defined as:

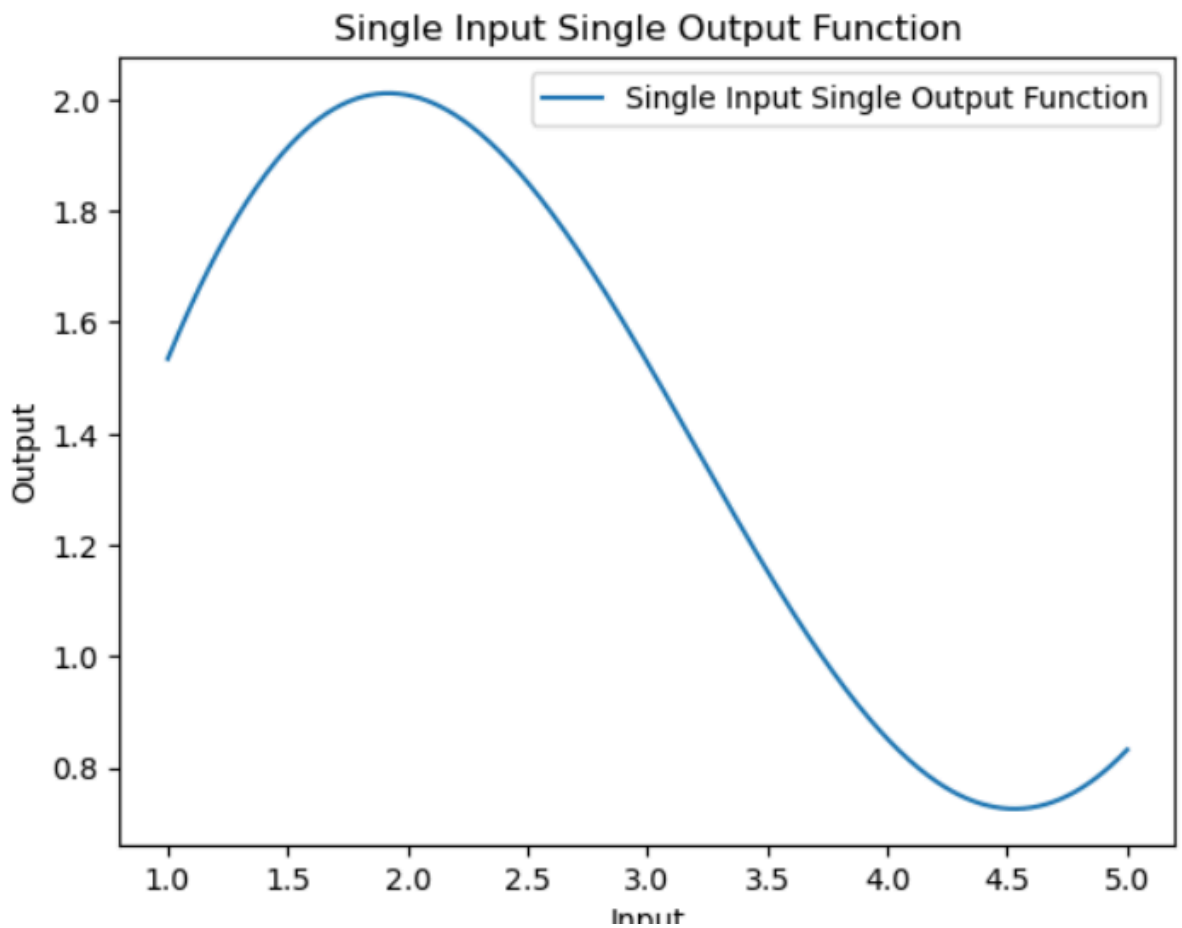
```
def target_function(x):
    return np.sin(x) + np.log(x + 1)

np.random.seed(0)
x_train = np.random.uniform(1, 5, 2000).reshape(-1, 1)
y_train = target_function(x_train).reshape(-1, 1)

x_val = np.linspace(1, 5, 400).reshape(-1, 1)
y_val = target_function(x_val).reshape(-1, 1)
```

This function takes a single input x and returns a combination of the sine of x and the logarithm of x+1 , creating a non-linear and complex pattern for the model to learn. The training and validation data are generated from this function, with input values uniformly distributed between 1 and 5 for the training set (x_train) and linearly spaced for the validation set (x_val).

A plot of the target function shows the relationship between the input and the output, providing a visual understanding of the function's behavior over the interval [1, 5]. This visualization is crucial for understanding the complexity of the function that the neural networks aim to approximate.



Deep Neural Network Models

Three different DNN models were implemented to approximate the target function. Each model varies in terms of hidden layers and neurons, designed to explore how these configurations affect learning and generalization.

Model 1: Configuration and Performance

- **Architecture**: 1 input layer, 7 hidden layers with 8 neurons each, and 1 output layer.
- **Number of Parameters**: 529.
- **Convergence**: Achieved at epoch 638 with a loss of approximately 0.000672.

Model 2: Configuration and Performance

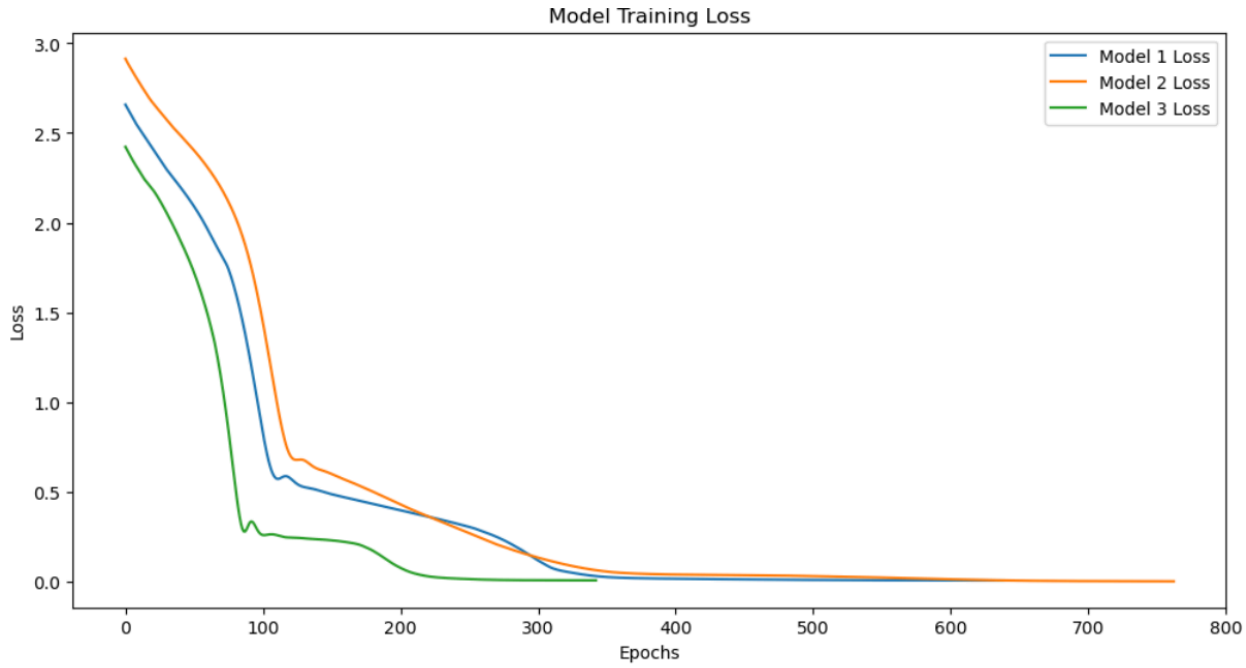
- **Architecture**: 1 input layer, 5 hidden layers with 10 neurons each, and 1 output layer.
- **Number of Parameters**: 581.
- **Convergence**: Achieved at epoch 762 with a loss of approximately 0.001414

Model 3: Configuration and Performance

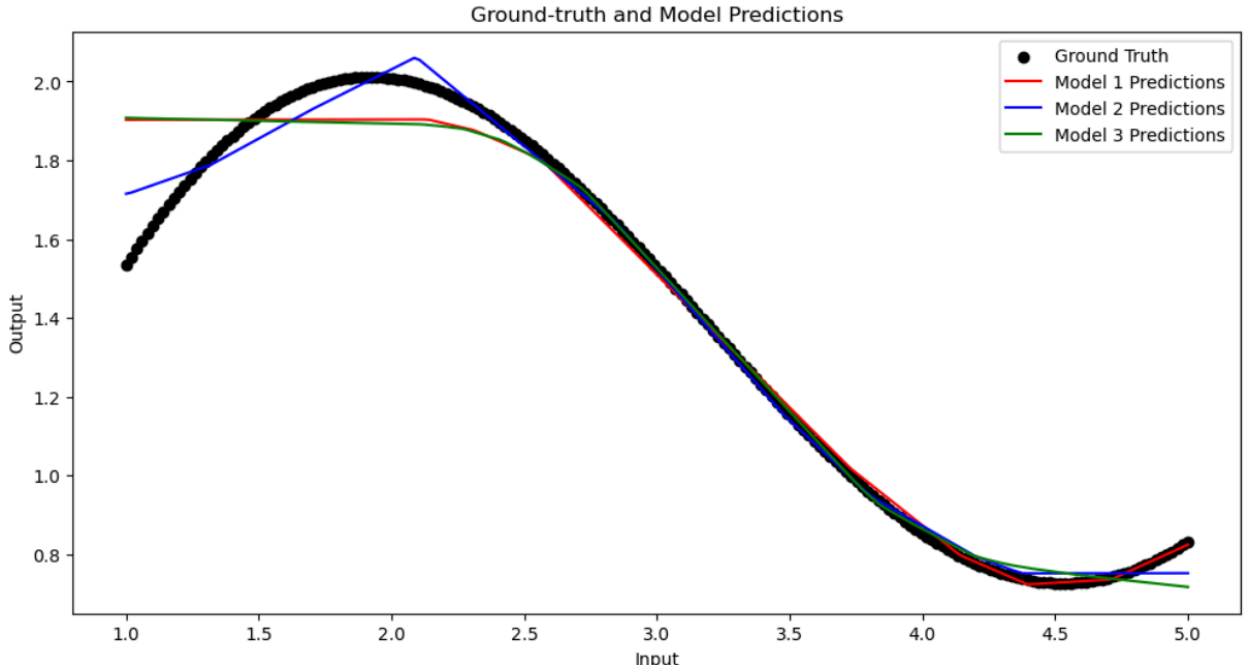
- **Architecture**: 1 input layer, 10 hidden layers with 14 neurons each, and 1 output layer.
- **Number of Parameters**: 2143.
- **Convergence**: Achieved at epoch 342 with a loss of approximately 0.006937.

Simulation of the function:

The models converged after reaching the epoch limit or when the model learns at a slow rate. Below is a graph showing the loss each of these models



The following graph shows the actual vs prediction for the 3 models.



Observation

The training of three neural network models demonstrates that while deeper models (like Model 3) converge faster due to their enhanced capacity to handle complex patterns, this does not always lead to the lowest loss. Model 2, with a moderate configuration, achieves better final accuracy, showing that optimal learning outcomes depend on a balanced model architecture rather than merely increasing depth and complexity.

FUNCTION 2

Target Function

The target function used in these experiments is a sinc function, defined as below.

This function takes a single input 5*x and returns a sine of 5*pi*x divided by 5*pi*x, creating a non-linear and complex pattern for the model to learn. The training and validation data are generated from this function, with input values

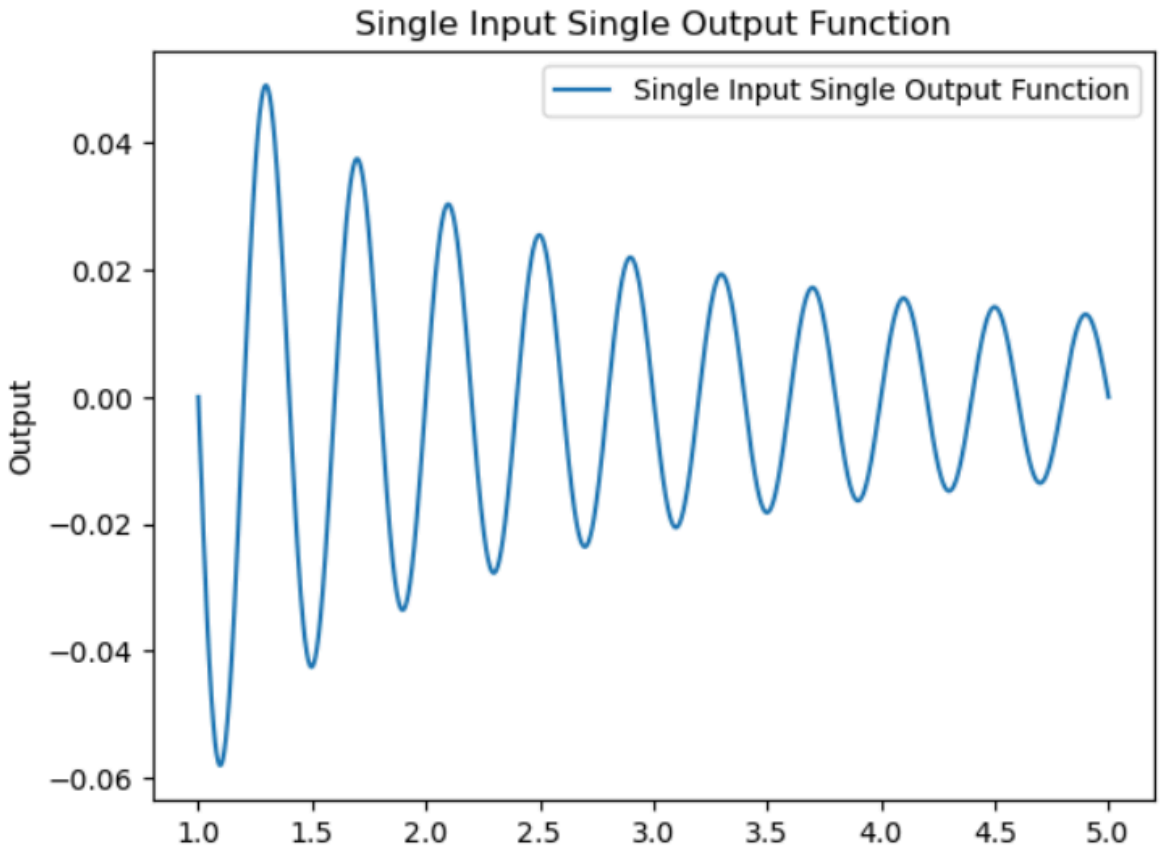
uniformly distributed between 1 and 5 for the training set (x_train) and linearly spaced for the validation set (x_val).

```
def target_function(x):
    return np.sinc(5*x)

np.random.seed(0)
x_train = np.random.uniform(1, 5, 10000).reshape(-1, 1)
y_train = target_function(x_train).reshape(-1, 1)

x_val = np.linspace(1, 5, 2000).reshape(-1, 1)
y_val = target_function(x_val).reshape(-1, 1)
```

A plot of the target function shows the relationship between the input and the output, providing a visual understanding of the function's behavior over the interval [1, 5]. This visualization is crucial for understanding the complexity of the function that the neural networks aim to approximate.



Deep Neural Network Models

Three different DNN models were implemented to approximate the target function. Each model varies in terms of hidden layers and neurons, designed to explore how these configurations affect learning and generalization.

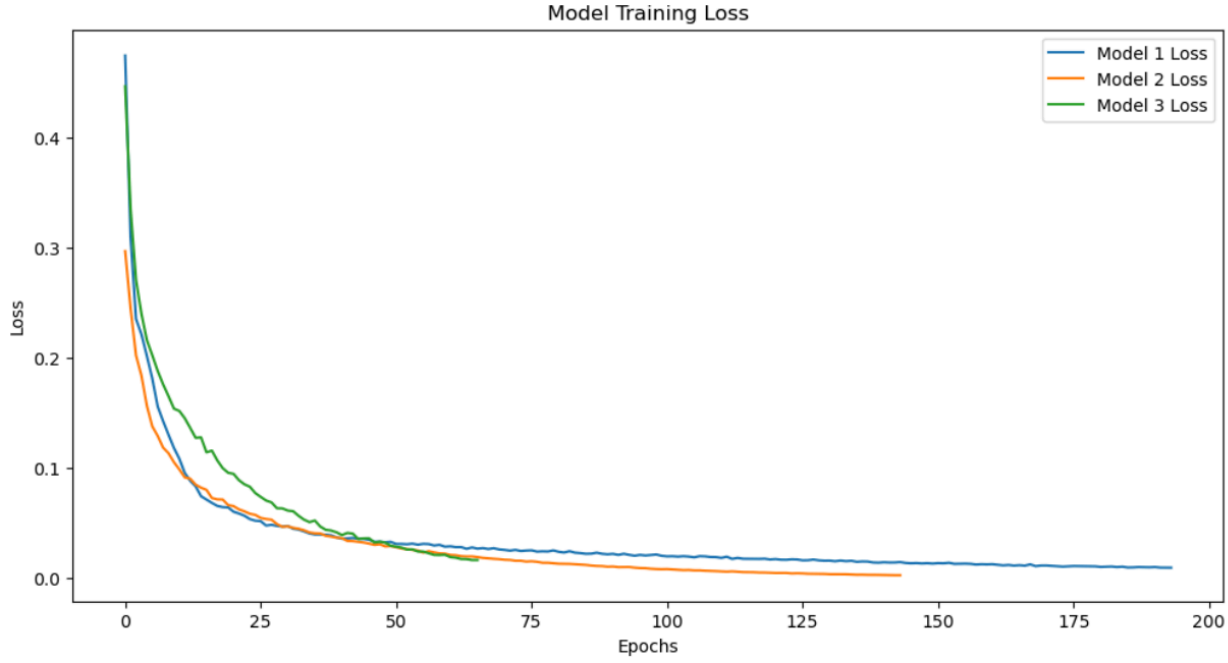
Model Configurations and Initial Observations

- **Model 1**: A less complex model with 50 hidden units per layer, 3 layers, and a dropout rate of 0.1. number of parameters 5551
- **Model 2**: A more complex model with 100 hidden units, 4 layers, and a dropout rate of 0.2. number of parameters 31401
- **Model 3**: The most complex model with 150 hidden units, 5 layers, and a dropout rate of 0.3. number of parameters 92551

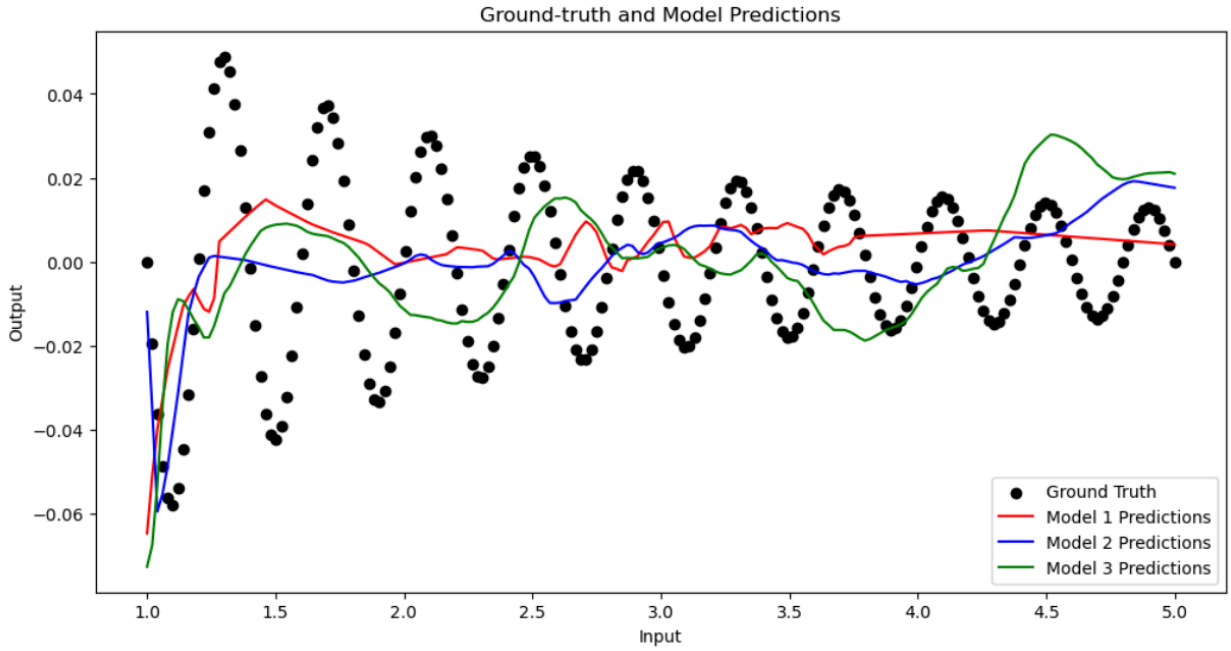
The complexity of these models increases not just in terms of the number of neurons but also layers and dropout rates, aiming to balance learning capacity with regularization to prevent overfitting.

Simulation of the function:

The models converged after reaching the epoch limit or when the model learns at a slow rate. Below is a graph showing the loss each of these models



The following graph shows the actual vs prediction for the 3 models.



Observation

Model 3, despite its rapid convergence, ended with a higher loss, demonstrating that increased complexity and regularization might not always equate to higher accuracy, especially with complex oscillatory targets. Model 2 had the lowest loss, slightly outperforming model 1. Model 3 failed to reach a low loss.

HW1_1_2 Train on Actual Tasks

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_1_2.ipynb

The three CNN models designed for MNIST image classification can be summarized by their key characteristics and hyperparameters as follows:

CNN Model 1

- Convolutional Layers: 2 layers with 10 and 20 filters, each with a kernel size of 5x5.
- Max-Pooling: Applied after each convolutional layer with a 2x2 window.
- Fully Connected Layers: Two layers with 120 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

CNN Model 2

- Convolutional Layers: 2 layers with 16 and 32 filters, each with a kernel size of 3x3 and padding=1.
- Max-Pooling: Applied after each convolutional layer with a 2x2 window.
- Fully Connected Layers: Two layers with 128 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

CNN Model 3

- Convolutional Layers: 2 layers with 8 and 16 filters, each with a kernel size of 3x3 and padding=1.
- Max-Pooling: Applied after each convolutional layer with a 2x2 window.
- Fully Connected Layers: Two layers with 64 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

Hyperparameters

• Learning Rate: 0.001

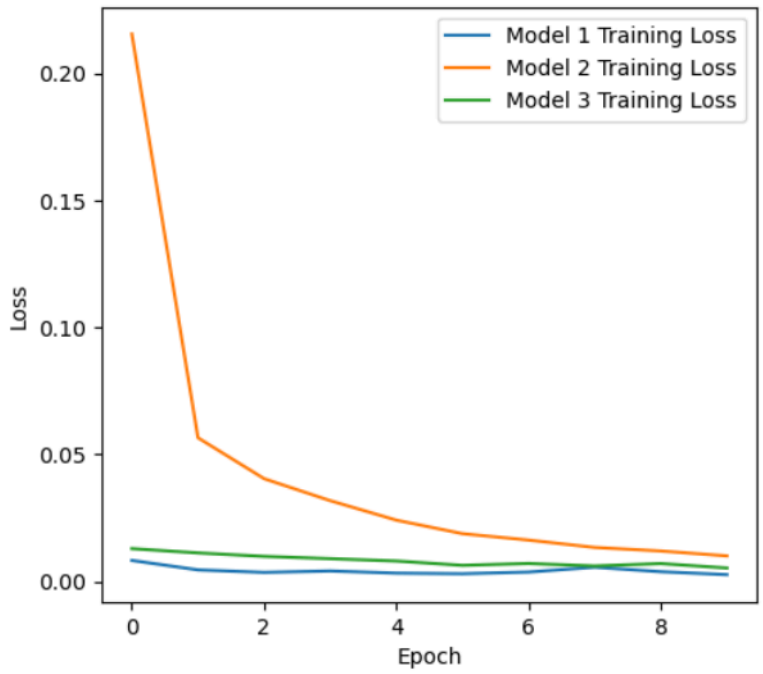
• Optimizer: Adam

• train_batch_size = 100

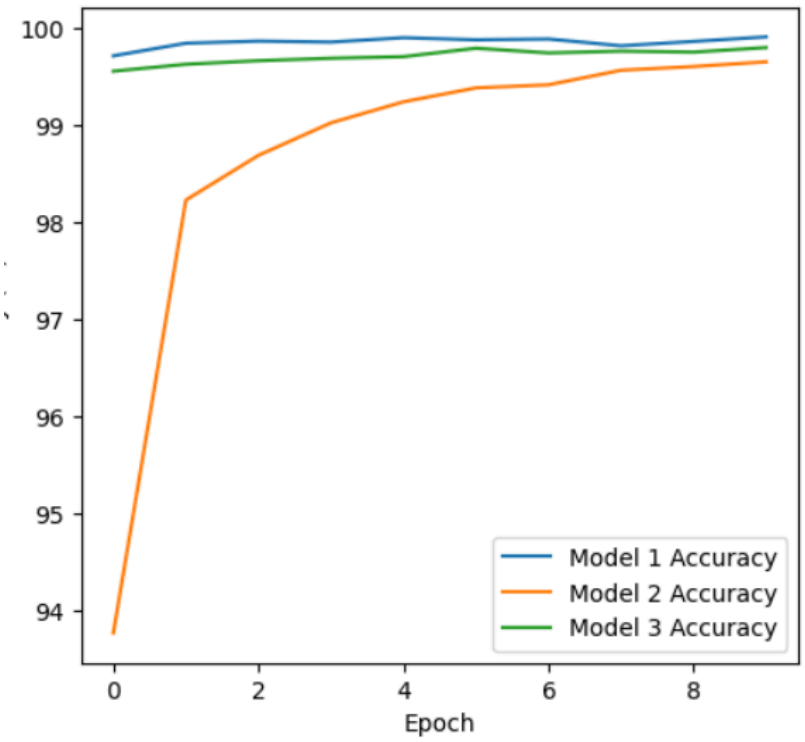
• test_batch_size = 64

• epochs = 10

• Loss: Cross Entropy



training loss for Model 1, Model 2 and Model 3



training accuracy for Model 1, Model 2 and Model 3

Observation

Model 2 excels in learning and generalizing on the MNIST dataset, improving significantly from high initial training loss to the lowest test loss, indicating a well-balanced model without overfitting. Meanwhile, Models 1 and 3, despite training improvements, show fluctuating test losses suggesting some overfitting.

HW 1-2 Optimization

The below model is trained on the MNIST dataset.

HW 1_2_1 Visualize the optimization process

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_2_1.ipynb

Neural Network Architecture (Net)

- A fully connected network with three layers: an input layer with 784 neurons (for MNIST images of size 28x28, flattened), a hidden layer with 128 neurons, a second hidden layer with 64 neurons, and an output layer with 10 neurons (corresponding to the 10 digit classes).
- The ReLU activation function is used after the first and second fully connected layers, with the final layer's output being raw scores.

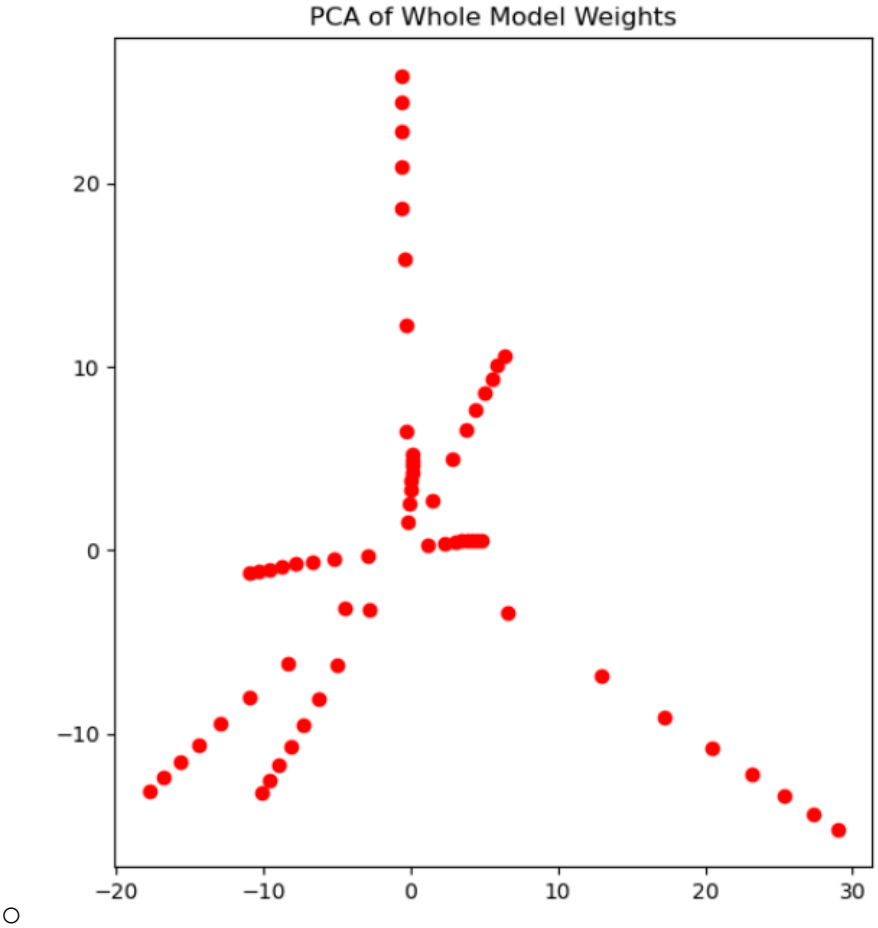
Training Setup

- Dataset: The MNIST dataset, normalized using a mean and standard deviation of 0.5. The dataset is loaded with a batch size of 64 for both training and testing, with shuffling enabled for the training set.
- Training Loop: The model is trained 8 times independently. Each training session spans 24 epochs, but weights are collected every 3 epochs, resulting in 8 collections per training session.
- Optimizer: The Adam optimizer is used with a learning rate of 0.001.
- Loss Function: CrossEntropyLoss is employed to compute the loss between the model's predictions and the true labels.

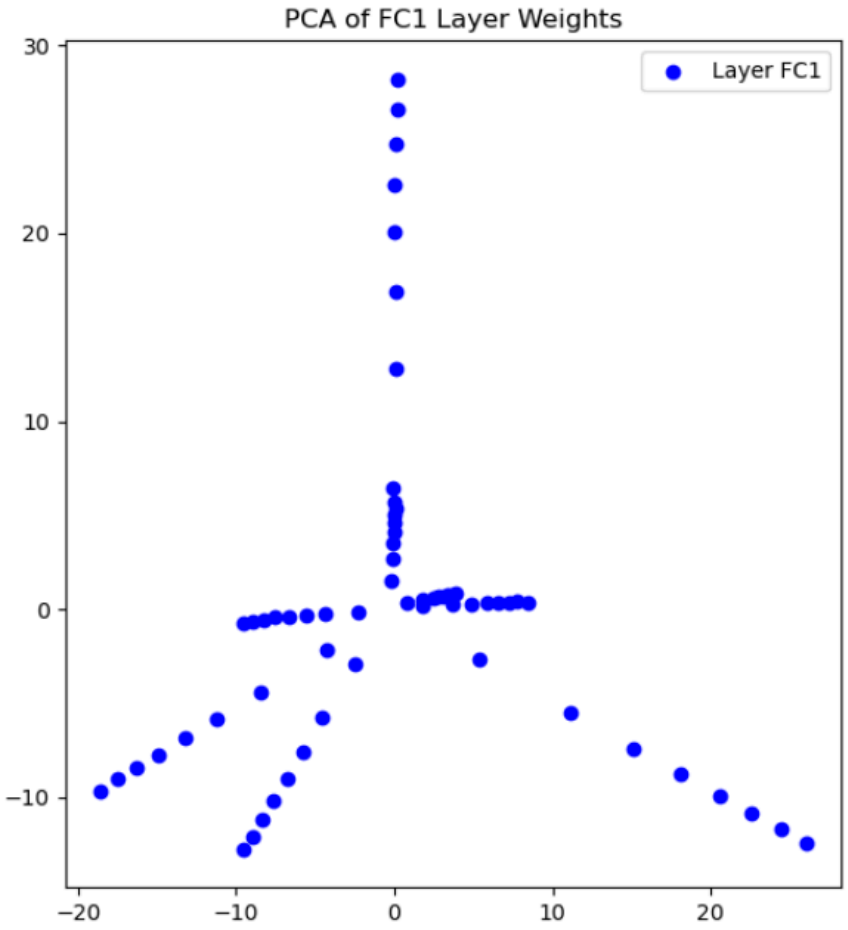
Weight Collection and Analysis

- Weight Collection: For each training run, the script collects the flattened weights of the first fully connected layer (fc1) and all layers combined at specified epochs.
- PCA Reduction: After training, PCA is applied to reduce the dimensionality of the collected weights to two principal components. This process is performed separately on the weights from the entire model and just the first fully connected layer (fc1)

Visualization

- Two scatter plots are generated:
    - The first plot visualizes the PCA-reduced weights from the entire model, aiming to observe the variance and clustering of the model's parameters through training.
    - 

- The second plot focuses on the PCA-reduced weights from the first fully connected layer (fc1), providing insights into how the parameters of this specific layer evolve.

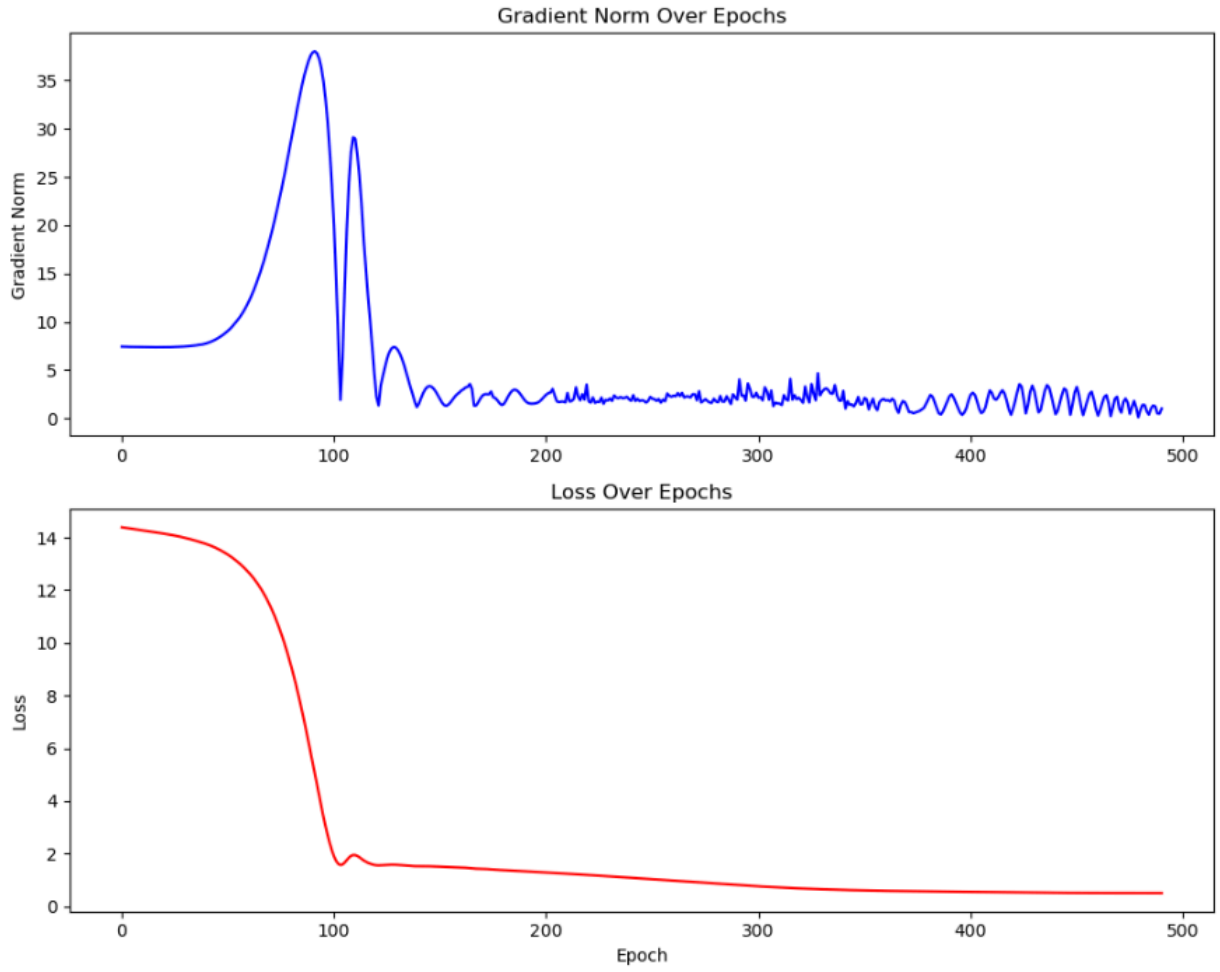


HW 1_2_2 Observe gradient norm during training

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_2_2.ipynb

The function sin(x)+log(x+1) function has been reused, which takes a single input x. The training and validation data are generated from this function, with input values uniformly distributed between 1 and 100 for the training set (x_train) and linearly spaced for the validation set (x_val).

Below is a graph for gradient norm, loss across the epochs



Observation

The neural network shows a promising learning trajectory with initial rapid gains as indicated by the sharp decrease in both gradient norms and loss, suggesting effective initial learning. As the training progresses, both metrics stabilize, implying that the network is nearing convergence with an optimal set of weights.

HW 1_2_3 What Happened When Gradient is Almost Zero

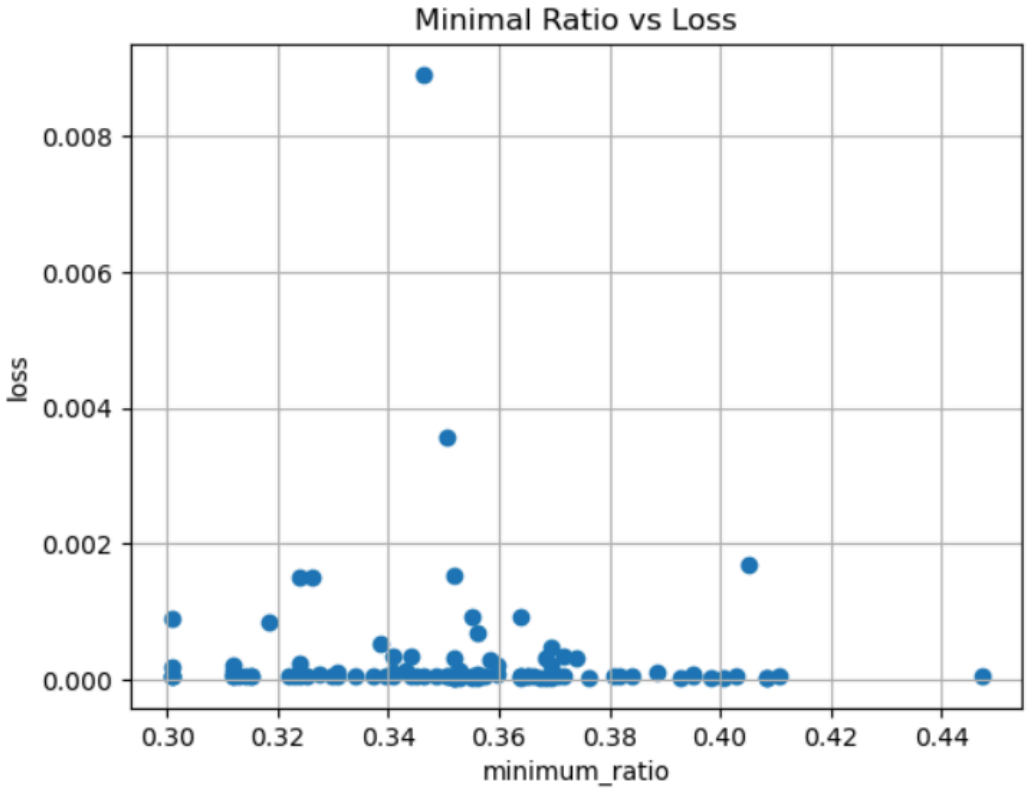
https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_2_3.ipynb

This function sinc(x)+log(x+1) function has been used as the target function

The minimal ratio represents the ratio of positive curvatures to the total curvatures in the loss landscape. It's calculated by assessing the second derivatives of the loss function with respect to each model parameter. A positive second derivative signifies positive curvature. The minimal ratio is obtained by dividing the count of positive curvature elements by the total count of curvature elements.

The **compute_minimal_ratio** function evaluates the trained model on the training data to compute the loss. It then iterates over each parameter to calculate the second derivatives, identifying positive curvatures. Accumulating these, it computes the minimal ratio, representing the proportion of positive curvatures to total curvatures. This ratio, along with the loss value, is returned for each iteration of training.

Below figure shows the plot for minimal ratio vs loss



HW 1-3 Generalization

The below model is trained on the MNIST dataset.

HW 1_3_1 Can network fit random labels?

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_3_1.ipynb

Model Architecture (CNN_Model)

- Convolutional Layers: Two convolutional layers are used. The first has 10 filters with a kernel size of 5, and the second has 20 filters with a kernel size of 5.
- Pooling Layers: Max pooling with a 2x2 window is applied after each convolutional layer.
- Fully Connected Layers: After flattening, the data pass through two fully connected layers. The first has 120 neurons, and the output layer has 10 neurons.
- Activation Functions: ReLU is used after each convolutional and the first fully connected layer. The output layer uses log softmax.

Dataset and Preprocessing

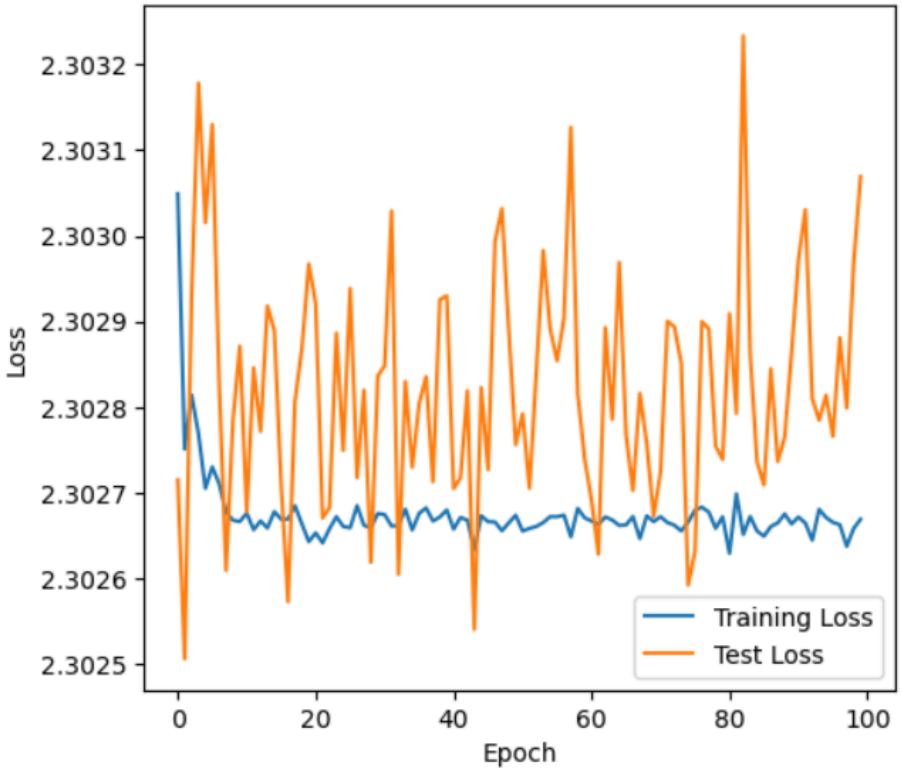
- MNIST Dataset: The standard MNIST dataset is used, resized to 32x32 pixels. The dataset is normalized using a mean of 0.1309 and a standard deviation of 0.3083.
- Train Set Randomization: The training set targets are replaced with random integers from 0 to 9, effectively randomizing the labels and making the task much more difficult, as the labels do not correspond to the actual images.

Training Process

- Batch Sizes: The batch size for training is 100, and for testing, it is 64.
- Data Loaders: PyTorch data loaders are used to load the train and test datasets with shuffling enabled for the training set.
- Training Loop: The model is trained for 100 epochs, recording the training loss at each epoch.

- Loss Function: CrossEntropyLoss is used.
- Optimizer: Adam optimizer is used with a learning rate of 1e3.
- Validation: After each training epoch, the model's performance is evaluated on the test set, and the test loss is recorded.

Below is the figure of the relationship between training and testing, loss and epochs.



Observation

There is some divergence between the training and test loss, which is a clear indicator that the model's learning is not generalizable. The network is likely overfitting to the noise (random labels) in the training data, which does not provide a meaningful signal for correct digit classification.

HW 1_3_2 Number of parameters vs Generalization

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_3_2.ipynb

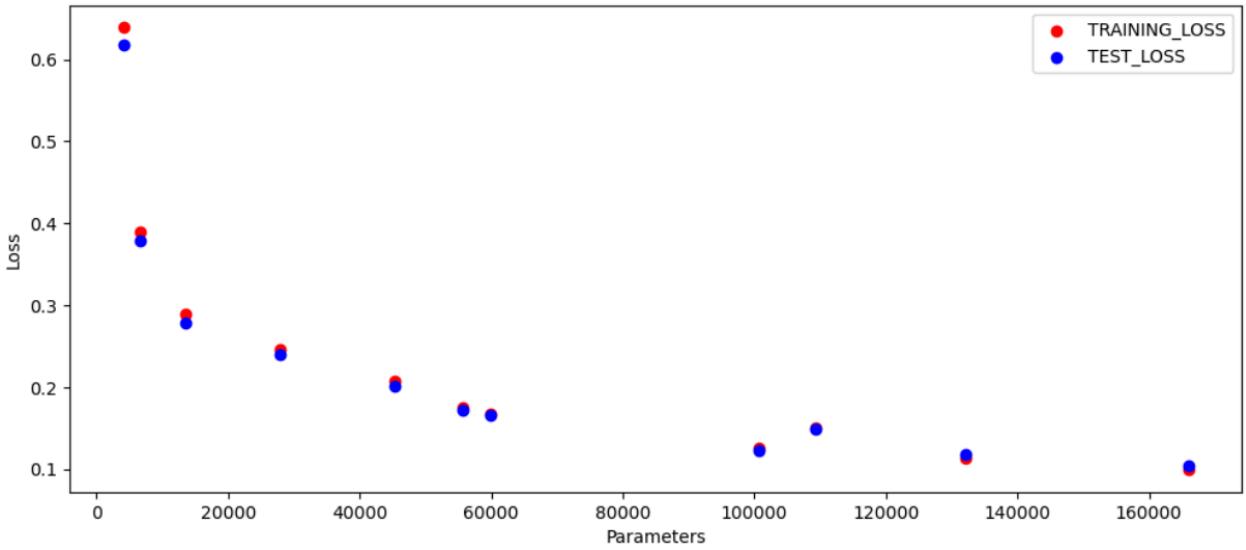
Models' Architecture:

- Models (Net_1 to Net_11) vary in the number of neurons in their layers:
  - Net_1: 128, 64 neurons
  - Net_2: 64, 128 neurons
  - Net_3: 32, 64 neurons
  - Net_4: 16, 32 neurons
  - Net_5: 8, 16 neurons
  - Net_6: 5, 9 neurons
  - Net_7: 50, 100 neurons
  - Net_8: 60, 120 neurons
  - Net_9: 100, 200 neurons
  - Net_10: 125, 250 neurons
  - Net_11: 150, 300 neurons

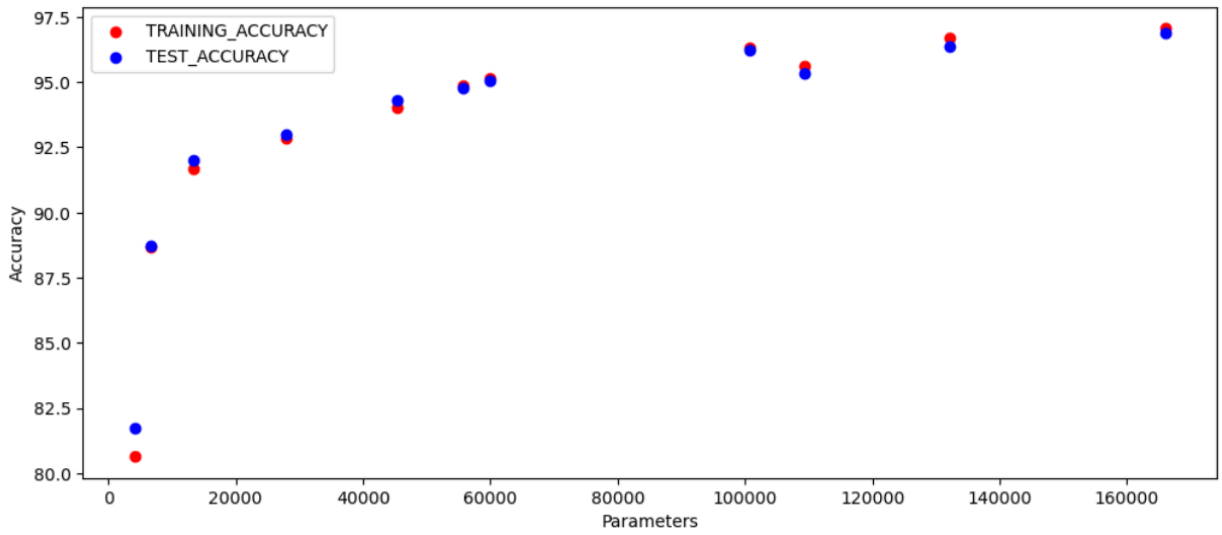
Training Settings:

- Optimizer: Adam with a learning rate of 0.0001.
- Loss Function: CrossEntropyLoss
- Epochs: Each model is trained for 10 epochs.
- Batch Size: 64 for both training and testing.
- Dataset Preprocessing: MNIST images are transformed to tensors and normalized with mean 0.5 and standard deviation 0.5.

Below is a figure of training and testing, loss to the number of parameters.



Graph for Accuracy comparison for the various models



Observation

The models are defined such that the number of parameters increase from Net_1 to Net_11 Similarly The plots indicate that models with more parameters initially show improved performance, with both accuracy increasing and loss decreasing. However, after reaching a certain number of parameters, the performance gains level off.

HW 1_3_3_1 Flatness vs Generalization part-1

https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_3_3_1.ipynb

The below model is trained on the MNIST dataset.

CNN Model Architecture (CNN_Model)

- Convolutional Layers: Two layers with 16 and 32 filters respectively, both using a kernel size of 3 and padding of 1.
- Fully Connected Layers: Two layers, with the first having 128 neurons and the second having 10 neurons, which corresponds to the number of classes in the MNIST dataset.
- Activation and Pooling: ReLU activation is used after each convolutional layer, followed by max-pooling with a window of size 2.

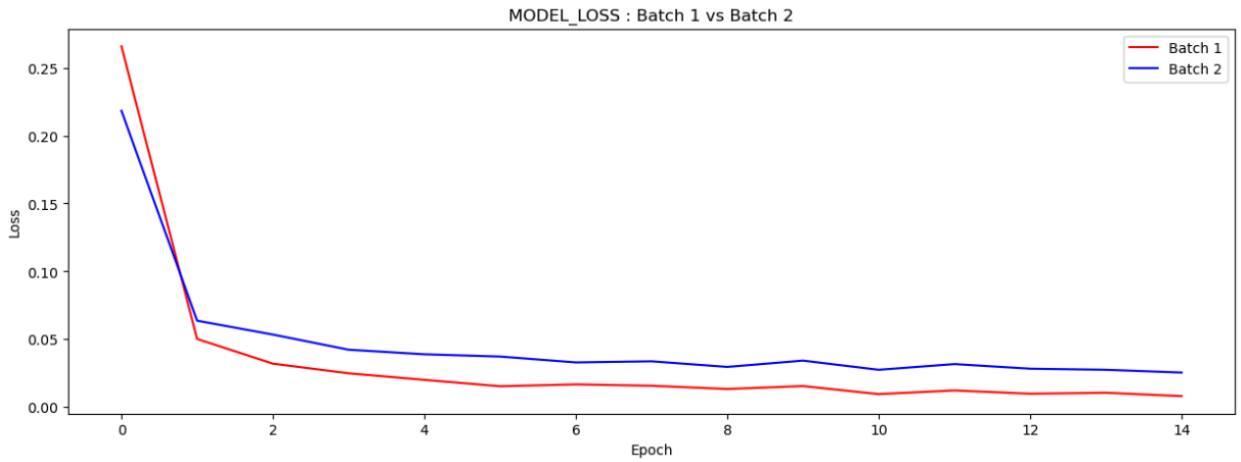
Training Approach

- Batch Sizes: Two separate trainings are conducted with batch sizes of 600 and 200.
- Loss Function: CrossEntropyLoss is used.
- Optimizer: Adam with a learning rate of 0.01.
- Training Duration: Each model is trained for 15 epochs.
- Data Preprocessing: MNIST images are resized to 32x32, converted to tensors, and normalized.

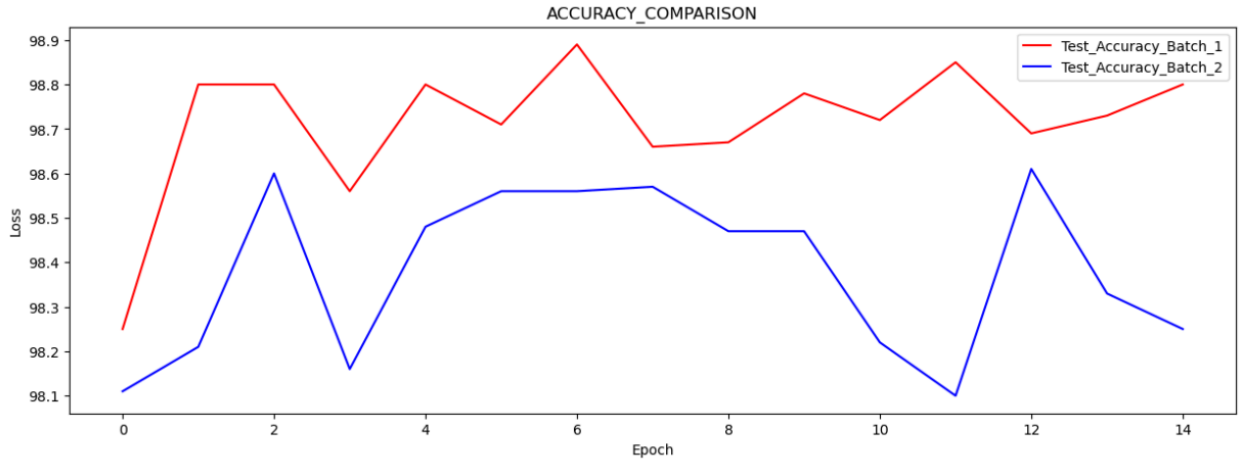
Parameter Space Analysis

- Interpolation: Parameters from both trained models are linearly interpolated, parameterized by an alpha value ranging from 0 to 1 in 100 steps.
- Loss and Accuracy Computation: For each interpolated set of parameters, the model's loss and accuracy on the training and test sets are evaluated.

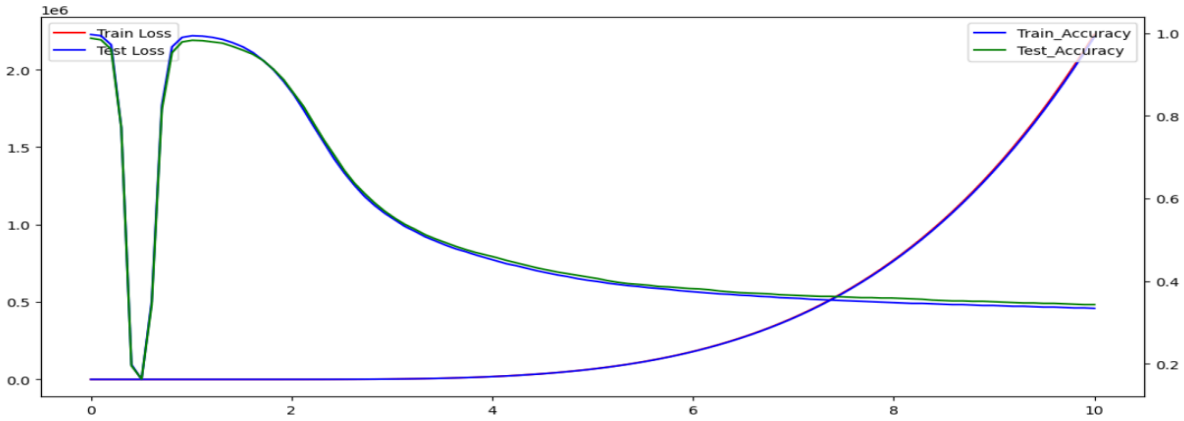
Below plots shows comparison of training loss for two models with two different batch sizes



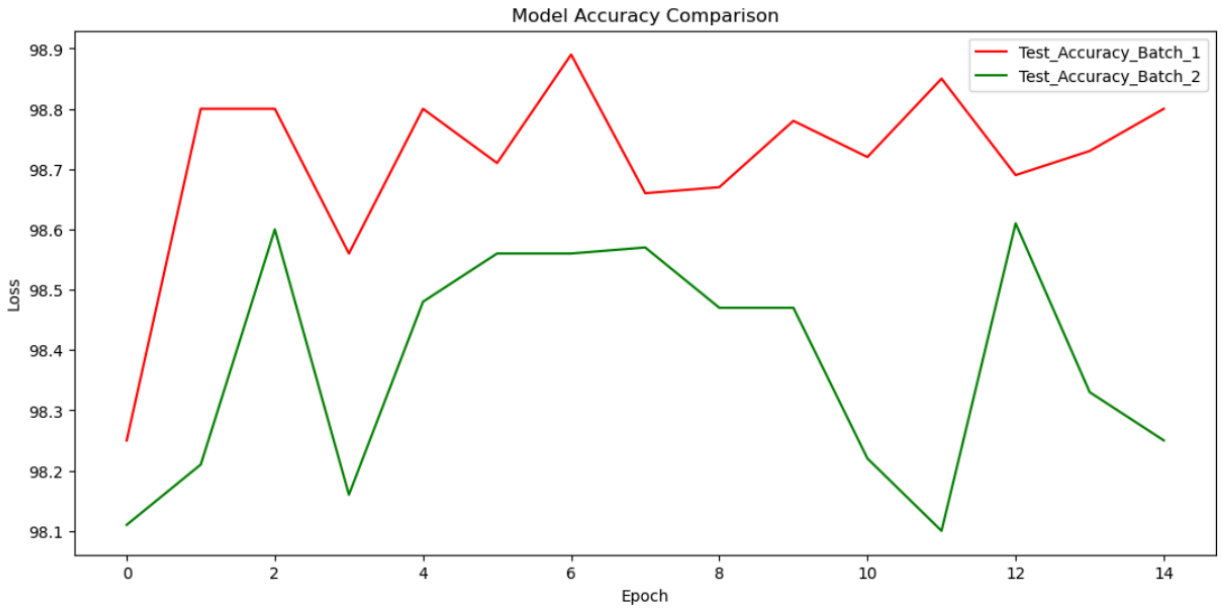
Below plots shows comparison of test accuracies for two models with two different batch sizes



Below plot shows the training and test loss and accuracy across the range of alpha values.



Below plot compares the testing accuracy of both original models over 15 epochs.



Learning Rate 1e-3

Observation

We can see the accuracy of test and train drop to around 0.75 for the models with different learning rates. The accuracy begins to increase for the graphs at alpha value of 4.

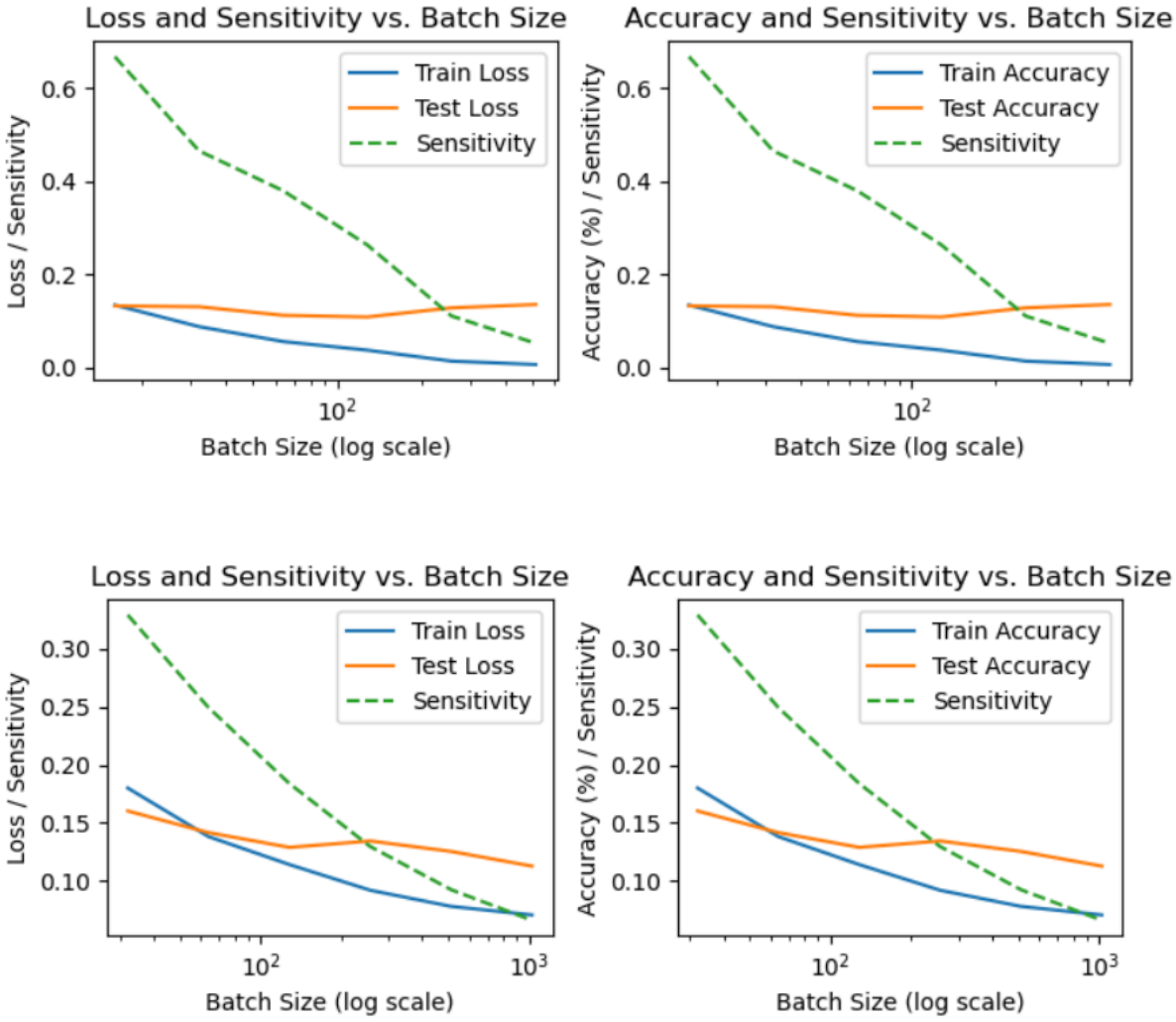
HW 1_3_3_2 Flatness vs Generalization part-2

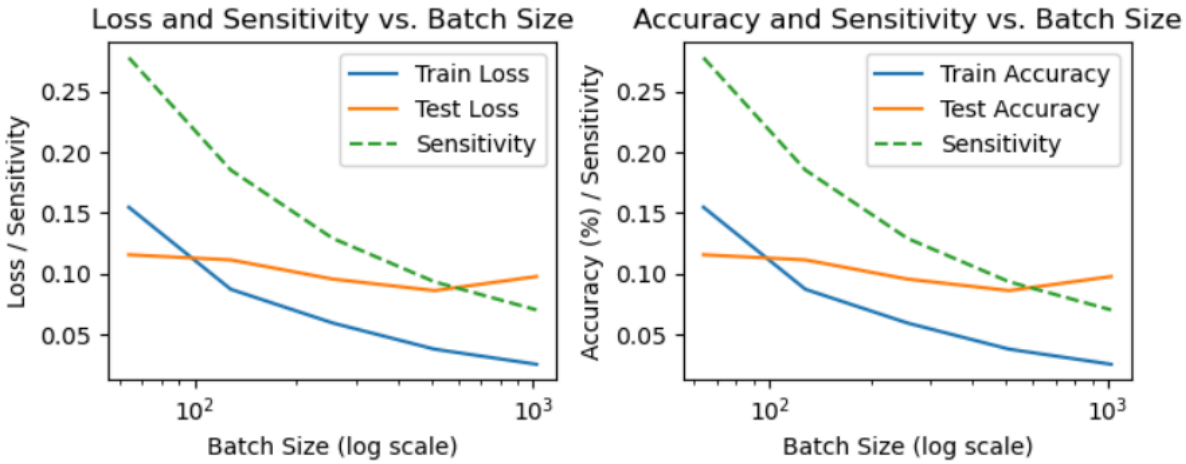
https://github.com/kalyan1998/DeepLearning/blob/main/ShivaKalyan_Diwakaruni_HW1_3_3_2.ipynb

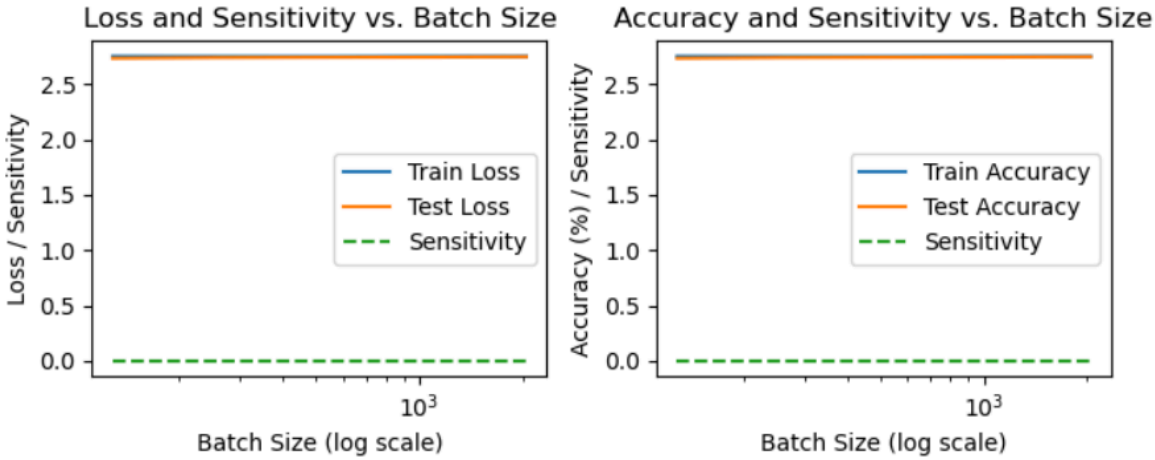
1. Network Structure: The CNN model consists of two convolutional layers followed by two fully connected layers. The first convolutional layer has 16 output channels, while the second has 32 output channels. ReLU activation functions are used after each convolutional layer, and max-pooling with a kernel size of 2 is applied after each activation. The fully connected layers have 128 and 10 units, respectively, with ReLU activation for the first and log-softmax for the second.

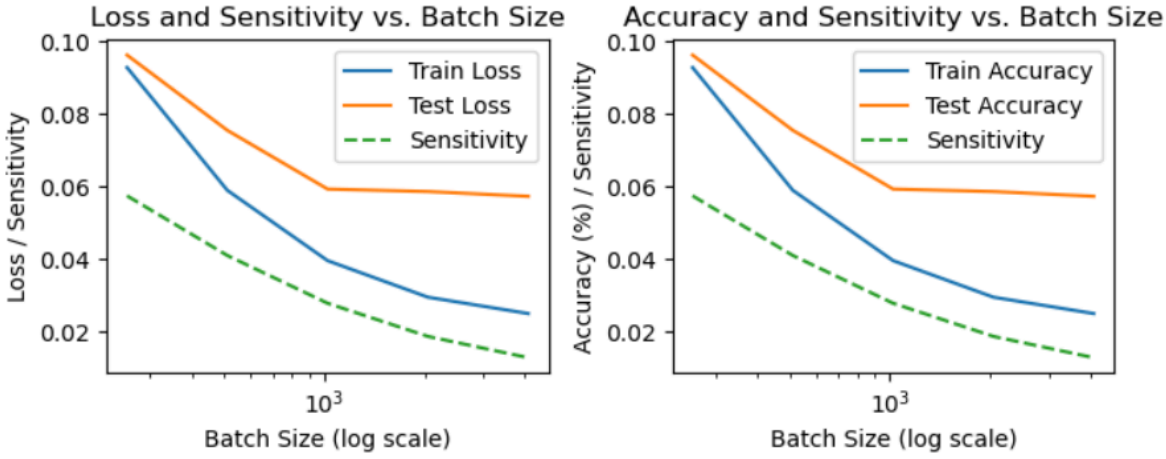
2. Training Approach: The model is trained using the Adam optimizer with a learning rate of 0.01. Cross-entropy loss is utilized as the loss function.
3. Batch Sizes: Experiments are conducted with various batch sizes ranging from 16 to 4096. For each batch size, the model is trained for a 5 epochs.
4. Data Preprocessing: The MNIST dataset is utilized for training and testing. the images are simply resized to 32x32 pixels and converted to tensors. Normalization is performed with a mean of 0.1309 and a standard deviation of 0.3083.

Below are figure of minimal ratio to the loss for 5 different models trained at 5 different batch sizes for each model




Loss and Sensitivity vs. Batch Size
Loss / Sensitivity
Batch Size (log scale)
Train Loss
Test Loss
Sensitivity
Accuracy and Sensitivity vs. Batch Size
Accuracy (%) / Sensitivity
Batch Size (log scale)
Train Accuracy
Test Accuracy
Sensitivity


Loss and Sensitivity vs. Batch Size
Loss / Sensitivity
Batch Size (log scale)
Train Loss
Test Loss
Sensitivity
Accuracy and Sensitivity vs. Batch Size
Accuracy (%) / Sensitivity
Batch Size (log scale)
Train Accuracy
Test Accuracy
Sensitivity


Loss and Sensitivity vs. Batch Size
Loss / Sensitivity
Batch Size (log scale)
Train Loss
Test Loss
Sensitivity
Accuracy and Sensitivity vs. Batch Size
Accuracy (%) / Sensitivity
Batch Size (log scale)
Train Accuracy
Test Accuracy
Sensitivity

Observation

As the batch size increased, sensitivity and training accuracy initially decreased while test accuracy remained relatively consistent, even though with some fluctuations. This trend persisted until reaching a batch size of 10^3 on a logarithmic scale, at which point all three metrics stabilized. However, beyond this point, sensitivity, training accuracy, and test accuracy all began to decline.